Due at 4:00pm on Tuesday 27 February 2018

What you need to get

- `YOU_a3.ipynb`: a Python notebook (hereafter called "the notebook")
- `Network.pyc`: Module with solution for `Network` class

What you need to know

This is a draft of the assignment, and it might be updated from time to time with clarifications – see the version number and release date at the bottom.

What to do

1. [5 marks] Consider the classification problem in which you have K classes, labelled $\{1, 2, \ldots, K\}$. Suppose we have a labelled dataset containing corresponding inputs and class labels, $(\vec{x}, \ell)$, where $\vec{x} \in \mathbb{R}^X$ and $\ell \in \{1, 2, \ldots, K\}$.

   Your neural network's output is a classification vector based on the softmax activation function, so that if z_k is the input current for output node k, then the activation of output node y_k is

   $$y_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}} \quad , \quad k = 1, \ldots, K \, .$$

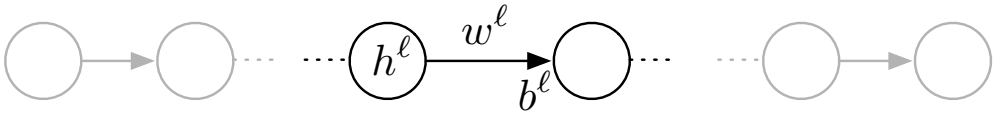
   Thus, $\vec{y} \in [0, 1]^K$, and $y_k = P\left(\ell = k \,|\, \vec{x}\right)$.

   Suppose that your loss function is cross entropy,

   $$E(\vec{y}, \vec{t}) = -\sum_{k=1}^{K} t_k \ln y_k \, .$$

   Derive an expression for $\frac{\partial E}{\partial z_j}$, the gradient of the loss function with respect to the input current to the output layer.

2. Consider a deep network that is simply a chain of nodes, as shown below.

   Assume that the activation function, $\sigma(\cdot)$, is the logistic function. Then the deep gradients for this network take the form,

   $$\frac{\partial E}{\partial z} = \prod_{\ell} w^\ell \sigma'(z^\ell) \quad \text{where } \; z^\ell = w^\ell h^\ell + b^\ell \, ,$$

   That is, while propagating the error gradients back through the network, each layer adds a term of the form $w^\ell \sigma'(z^\ell)$. Dropping the superscripts, consider the magnitude of one generic term in that product, $|w\sigma'(wh + b)|$.

   (a) [1 mark] Suppose $|w\sigma'(wh + b)| \geq 1$. Prove that this can only occur if $|w| \geq 4$.

(b) [5 marks] Supposing that $|w| \geq 4$, consider the set of input currents h for which $|w\sigma'(wh+b)| \geq 1$. Show that the set of activity values for h satisfying that constraint can range over an interval no greater in width than

$$\frac{2}{|w|} \ln \left[\frac{|w|}{2} \left(1 + \sqrt{1 - \frac{4}{|w|}} \right) - 1 \right] .$$

(c) [3 marks] Plot the expression above over a range of w values and show that it peaks around $|w| = 6.9$.

3. **Implementing Dropout** [11 marks]

Recall that the supplied `Network` module has a `Network` class, and includes an implementation of error backpropagation. For this question, you will extend the `Network` class, and create a class called `RobustNetwork`. In this class, you will re-implement the function `FeedForward` so that it can also implement dropout with a 50% dropout ratio with a call such as

```
net.FeedForward(x, dropout=True)
```

Note that `Network.Learn` has an additional – previously unused – argument, `dropout`. By default, `dropout` is `False`. But, calls such as

```
net.Learn(train, epochs=10, lrate=0.005, dropout=True)
```

will add the argument `dropout=True` to calls to `FeedForward`, like that shown above.

For your convenience, the notebook includes a sample regression dataset for you to test your implementation on. Your implementation should work with these lines of code. Also, a non-dropout version of `FeedForward` is supplied below.

(a) Complete the implementation of `FeedForward` in the `RobustNetwork` class to implement dropout as described above. Your implementation of `FeedForward` must include the optional `dropout` parameter, and needs to revert the non-dropout behaviour when `dropout` is `False` (or absent). When `dropout=True`, half of the hidden nodes (randomly selected) should be set to zero. Note that you should then temporarily double the connection weights to compensate for the missing contribution from the removed nodes.

(b) Create two identical networks, one called `original_net` and one called `robust_net`. You can use `copy.deepcopy` to do that.

(c) Train `original_net` on the `train` dataset using the `Learn` function with 3000 epochs, a learning rate of 0.0005, and `dropout=False`. Evaluate the network on the training and test datasets.

(d) Train `robust_net` on the `train` dataset using the same parameters as above, but with `dropout=True`. Again, evaluate the network on the training and test datasets.

(e) Plot the original training points (blue dots), the test points (yellow), as well as lines showing the models learned by `original_net` (blue dashed line) and `robust_net` (red dashed line). As always, label the axes.

```python
def FeedForward(self, x):
    '''
        y = net.FeedForward(x)

        Runs the network forward, starting with x as input.
        Returns the activity of the output layer.

        Note: The activation function used for hidden layers depends
        on what self.Activation is set to. And the activation function for
        the output layer depends on what self.OutputActivation is set to.
    '''
    self.h[0] = x # set input layer

    # For each layer...
    for l in range(1, self.n_layers):
        # Calc. input current to next layer
        z = np.dot(self.W[l-1], self.h[l-1]) + self.b[l]

        # Use activation function for hidden nodes.
        # For output layer, use chosen activation.
        if l<self.n_layers-1:
            self.h[l] = self.Activation(z)
        else:
            # For a regression network, use linear activation
            # for output layer.
            self.h[l] = self.OutputActivation(z)
    # Return activity of output layer
    return self.h[-1]
```

4. **Deep Networks** [6 marks]

   The notebook has a dataset that classifies 2-dimensional points into three distinct categories. Your task is to devise a neural-network architecture that will yield high accuracy in this classification task.

   Here's the catch: You have a fixed computational budget. You are allowed to use 40 hidden nodes, and you can place them into layers as you see fit. The input layer needs 2 nodes, and the output layer has 3 nodes. But, for example, you can have a single hidden layer of 40 nodes, or three hidden layers of 15, 10, and 15. It's up to you. Find a network configuration that works well. What gives the best training and testing accuracy?

   (a) Create a network and evaluate it on the training and test datasets.

   (b) Train your network on the training data using one call of

   ```
   progress = net.SGD(train, epochs=500, lrate=0.5)
   ```

   (c) Evaluate your trained network on the training and test datasets.

   (d) Plot the original points, but coloured according to your network's one-hot classification. Make one plot for the training data, and one for the test data.

Enjoy!

What to submit

Your assignment submission should be a single jupyter notebook file, named (`<WatIAM>_a3.ipynb`), where `<WatIAM>` is your UW WatIAM login ID (not your student number). The notebook must include solutions to **all** the questions. Submit this file to Desire2Learn. You do not need to submit any of the modules supplied for the assignment.